Inteligência Computacional

PROJETO – FASE3

Elementos do grupo:
Luís Henrique P. O. Travassos, nº2021136600
Rodrigo Ramalho Ferreira, nº2021139149

isec Engenharia

Index

Descrição das Metodologias usadas

- **Otimização por PSO (Particle Swarm Optimization):**
  - Utilizou a técnica de otimização por enxame de partículas.
  - Aplicado a um conjunto mais reduzido de dados (50 imagens por classe) para experimentação rápida.
- **Otimização por Random Search:**
  - Focou na utilização da busca aleatória para a otimização de hiperparâmetros.
  - Beneficiou-se do mesmo conjunto de dados reduzido para agilizar o processo.
- **Otimização por Grid Search:**
  - Um terceiro programa utilizou a técnica de Grid Search, uma abordagem mais sistemática e exaustiva.
  - Este método também foi testado no conjunto de dados reduzido.
- **Modelo Simples (Simple):**
  - Adotou uma abordagem direta sem técnicas de otimização.
  - Utilizou o melhor conjunto de hiperparâmetros identificados nas três metodologias anteriores.
  - Este programa foi treinado num conjunto substancialmente maior (4324 imagens) para avaliar a escalabilidade e o desempenho.
- **Modelo Para Uso do Utilizador (ToUse):**
  - O quinto programa foi projetado para aplicar o modelo treinado no 'Simple' a um conjunto de dados personalizado fornecido pelo utilizador.
  - Este conjunto é composto por um número limitado de imagens escolhidas pelo utilizador, permitindo uma análise personalizada e específica.

ESTRUTURA DE CÓDIGO DO MODELO SIMPLES

- *Importação das bibliotecas*
- *Tratamento das Imagens*
- *Definição, compilação e treinamento*
- *Avaliação e análise*
- *Armazenamento do Excel e Modelo*

```python
# Importação das bibliotecas
from scipy.stats import loguniform, uniform
import numpy as np
import pandas as pd
import os

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, Input
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.applications import VGG16
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score
from sklearn.preprocessing import LabelEncoder, label_binarize
from sklearn.utils import shuffle
from sklearn.model_selection import RandomizedSearchCV

from SwarmPackagePy import gwo
import pyswarms as ps
```

```python
# Tratamento das imagens
def create_street_data(path, street_types, im_size):
    images, labels = [], []
    streets = [(item, os.path.join(path, item, street))
               for item in street_types
               for street in os.listdir(os.path.join(path, item))]
    streets_df = pd.DataFrame(streets, columns=['street type', 'image'])

    for _, row in streets_df.iterrows():
        img = load_img(row['image'], target_size=(im_size, im_size))
        images.append(img_to_array(img))
        labels.append(row['street type'])

    return np.array(images, dtype='float32') / 255.0, np.array(labels)
```

```python
im_size = 224

street_types = ['clean', 'litter', 'recycle']
path = '../Dataset/'
path_test = '../Dataset_Test/'
```

```python
train_images, train_labels = create_street_data(path, street_types, im_size)
test_images, test_labels = create_street_data(path_test, street_types, im_size)

streets_count = pd.value_counts(train_labels)
print("Streets in each category:", streets_count)
```

```python
label_encoder = LabelEncoder()
train_labels_encoded = label_encoder.fit_transform(train_labels)
test_labels_encoded = label_encoder.transform(test_labels)

train_x, val_x, train_y, val_y = train_test_split(train_images, train_labels_encoded, test_size=0.15, random_state=415)

print(f"Train shape: {train_x.shape}")
print(f"Validation shape: {val_x.shape}")
print(f"Test shape: {test_images.shape}")
```

```python
# Definição, compilação e treinamento
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(im_size, im_size, 3))

for layer in base_model.layers:
    layer.trainable = False

x = base_model.output
x = Flatten()(x)
x = Dropout(0.18)(x)
x = Dense(len(street_types), activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=x)

model.compile(optimizer=Adam(0.000495),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_x,
                    train_y,
                    epochs=5,
                    validation_data=(val_x, val_y))
```

```python
# Avaliação e análise
test_loss, test_accuracy = model.evaluate(test_images, test_labels_encoded, verbose=1)
print(f"\nAcurácia no teste: {test_accuracy*100:.2f}%\n")

y_pred_test = model.predict(test_images)
y_pred_test_classes = np.argmax(y_pred_test, axis=1)

confusion_mtx = confusion_matrix(test_labels_encoded, y_pred_test_classes)
print("Matriz de Confusão:")
print(confusion_mtx)

class_report = classification_report(test_labels_encoded, y_pred_test_classes, target_names=label_encoder.classes_)
print("\nRelatório de Classificação:")
print(class_report)

y_true_binarized = to_categorical(test_labels_encoded)

for i in range(y_true_binarized.shape[1]):
    auc_score = roc_auc_score(y_true_binarized[:, i], y_pred_test[:, i])
    print(f'AUC para a classe {label_encoder.classes_[i]}: {auc_score:.2f}')
```

```python
# Armazenamento do Excel e Modelo
shapes_data = {
    'Conjunto': ['Treino', 'Validação', 'Teste'],
    'Formato': [train_x.shape, val_x.shape, test_images.shape]
}
df_shapes = pd.DataFrame(shapes_data)

df_confusion_mtx = pd.DataFrame(confusion_mtx, index=label_encoder.classes_, columns=label_encoder.classes_)

report_data = classification_report(test_labels_encoded, y_pred_test_classes, target_names=label_encoder.classes_, output_dict=True)
df_class_report = pd.DataFrame(report_data).transpose()

auc_scores = {label_encoder.classes_[i]: roc_auc_score(y_true_binarized[:, i], y_pred_test[:, i]) for i in range(y_true_binarized.shape[1])}
df_auc_scores = pd.DataFrame(list(auc_scores.items()), columns=['Classe', 'AUC'])

with pd.ExcelWriter('IC_Project_Fase3_Simple_Excel.xlsx') as writer:
    df_shapes.to_excel(writer, sheet_name='Formas dos Conjuntos', index=False)
    df_confusion_mtx.to_excel(writer, sheet_name='Matriz de Confusão')
    df_class_report.to_excel(writer, sheet_name='Relatório de Classificação')
    df_auc_scores.to_excel(writer, sheet_name='AUC por Classe', index=False)

print("Resultados exportados para o arquivo 'IC_Project_Fase3_Simple_Excel.xlsx'")
```

```
Resultados exportados para o arquivo 'IC_Project_Fase3_Simple_Excel.xlsx'
```

```python
model_save_path = 'IC_Project_Fase3_Simple_SaveModel.h5'

model.save(model_save_path)
print(f"Modelo salvo com sucesso em: {model_save_path}")
```

```
Modelo salvo com sucesso em: IC_Project_Fase3_Simple_SaveModel.h5
```

ESTRUTURA DE CÓDIGO DO MODELO PSO

- *Definição, compilação e treinamento*
- *Implementação do PSO*
  - *Train_top_layer*
  - *Fitness_function*
  - *PSO*
- *Final Model*

```python
# Definição, compilação e treinamento
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(im_size, im_size, 3))

for layer in base_model.layers:
    layer.trainable = False
```

```python
# Implementação do PSO
def train_top_layer(hyperparameters, train_x, train_y, val_x, val_y, base_model):
    learning_rate, dropout_rate = hyperparameters

    top_model = Sequential([
        Flatten(input_shape=base_model.output_shape[1:]),
        Dropout(dropout_rate),
        Dense(3, activation='softmax')
    ])

    model = Model(inputs=base_model.input, outputs=top_model(base_model.output))

    model.compile(optimizer=Adam(learning_rate),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    history = model.fit(train_x,
                        train_y,
                        epochs=5,
                        validation_data=(val_x, val_y),
                        verbose=0)

    validation_loss, validation_accuracy = model.evaluate(val_x, val_y, verbose=0)
    return validation_loss
```

```python
def fitness_function(x, train_x, train_y, val_x, val_y, base_model):

    n_particles = x.shape[0]

    losses = []

    for i in range(n_particles):
        # x[i] contém os hiperparâmetros para a i-ésima partícula
        hyperparameters = x[i]
        loss = train_top_layer(hyperparameters, train_x, train_y, val_x, val_y, base_model)
        losses.append(loss)
        results.append({'Hiperparâmetros': hyperparameters, 'Perda': loss})

    return np.array(losses)
```

```python
results = []

bounds = [(0.0001, 0.1), (0.0, 0.5)]
options = {'c1': 0.5, 'c2': 0.3, 'w': 0.9}

optimizer = ps.single.GlobalBestPSO(n_particles=5,
                                    dimensions=2,
                                    options=options,
                                    bounds=bounds)

cost, best_pos = optimizer.optimize(fitness_function,
                                    iters=5,
                                    train_x=train_x,
                                    train_y=train_y,
                                    val_x=val_x,
                                    val_y=val_y,
                                    base_model=base_model)

best_learning_rate, best_dropout_rate = best_pos
```

ESTRUTURA DE CÓDIGO DO MODELO RANDOM SEARCH

- *Definição, compilação e treinamento*
- *Implementação do Random Search*
  - *Create_model*
  - *KerasClassifier*
  - *RandomizedSearch*
- *Final Model*

```
# Definição, compilação e treinamento
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(im_size, im_size, 3))

for layer in base_model.layers:
    layer.trainable = False
```

```
# Implementação da Random Search
def create_model(learning_rate, dropout_rate):
    top_model = Sequential([
        Flatten(input_shape=base_model.output_shape[1:]),
        Dropout(dropout_rate),
        Dense(3, activation='softmax')
    ])

    model = Model(inputs=base_model.input, outputs=top_model(base_model.output))

    model.compile(optimizer=Adam(learning_rate=learning_rate),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

```
model = KerasClassifier(build_fn=create_model, epochs=5, verbose=0)

param_distributions = {
    'learning_rate': loguniform(1e-5, 0.1),
    'dropout_rate': uniform(0.0, 0.5)
}
```

```
random_search = RandomizedSearchCV(estimator=model,
                                   param_distributions=param_distributions,
                                   n_iter=10,
                                   n_jobs=1,
                                   cv=3)

random_search_result = random_search.fit(train_x, train_y)

results = []
for params, mean_test_score in zip(random_search_result.cv_results_['params'], random_search_result.cv_results_['mean_test_score']):
    results.append({'Hiperparâmetros': params, 'Perda': 1 - mean_test_score})

best_learning_rate = random_search_result.best_params_['learning_rate']
best_dropout_rate = random_search_result.best_params_['dropout_rate']
print("Melhor: %f usando %s" % (random_search_result.best_score_, random_search_result.best_params_))
```

ESTRUTURA DE CÓDIGO DO MODELO GRID

- *Definição, compilação e treinamento*
- *Implementação do Random Search*
  - *Create_model*
  - *KerasClassifier*
  - *GridSearch*
- *Final Model*

```
# Definição, compilação e treinamento
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(im_size, im_size, 3))

for layer in base_model.layers:
    layer.trainable = False
```

```
# Implementação da Grid Search
def create_model(learning_rate, dropout_rate):
    top_model = Sequential([
        Flatten(input_shape=base_model.output_shape[1:]),
        Dropout(dropout_rate),
        Dense(3, activation='softmax')
    ])

    model = Model(inputs=base_model.input, outputs=top_model(base_model.output))

    model.compile(optimizer=Adam(learning_rate=learning_rate),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model
```

```
model = KerasClassifier(build_fn=create_model, epochs=5, verbose=0)

param_grid = {
    'learning_rate': [0.00001, 0.0001, 0.001, 0.01, 0.1],
    'dropout_rate': [0.0, 0.1, 0.25, 0.4, 0.5]
}
```

```
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=1, cv=3)

grid_result = grid.fit(train_x, train_y)

results = []

for params, mean_test_score in zip(grid_result.cv_results_['params'], grid_result.cv_results_['mean_test_score']):
    results.append({'Hiperparâmetros': params, 'Perda': 1 - mean_test_score})

best_learning_rate = grid_result.best_params_['learning_rate']
best_dropout_rate = grid_result.best_params_['dropout_rate']
print("Melhor: %f usando %s" % (grid_result.best_score_, grid_result.best_params_))
```

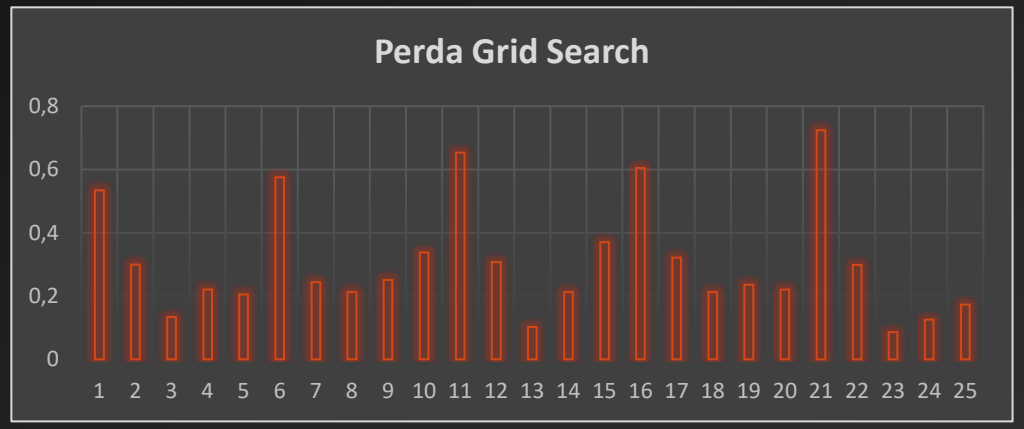
Análise de Resultados - Grid Search

Hiperparâmetros Investigados:

Taxas de aprendizado: [0.00001, 0.0001, 0.001, 0.01, 0.1];
Taxas de dropout: [0.0, 0.1, 0.25, 0.4, 0.5];
25 iterações;

Conclusões:

Accuracy: 84.18%
Melhor taxa de aprendizado: 0.001
Melhor taxa de dropout: 0.5



Numero	Hiperparâmetros	Perda
1	{'dropout_rate': 0.0, 'learning_rate': 1e-05}	0,53433
2	{'dropout_rate': 0.0, 'learning_rate': 0.0001}	0,299187899
3	{'dropout_rate': 0.0, 'learning_rate': 0.001}	0,133628647
4	{'dropout_rate': 0.0, 'learning_rate': 0.01}	0,221114794
5	{'dropout_rate': 0.0, 'learning_rate': 0.1}	0,205241799
6	{'dropout_rate': 0.1, 'learning_rate': 1e-05}	0,57567367
7	{'dropout_rate': 0.1, 'learning_rate': 0.0001}	0,243816912
8	{'dropout_rate': 0.1, 'learning_rate': 0.001}	0,212255458
9	{'dropout_rate': 0.1, 'learning_rate': 0.01}	0,251199702
10	{'dropout_rate': 0.1, 'learning_rate': 0.1}	0,338316719
11	{'dropout_rate': 0.25, 'learning_rate': 1e-05}	0,653008481
12	{'dropout_rate': 0.25, 'learning_rate': 0.0001}	0,307862679
13	{'dropout_rate': 0.25, 'learning_rate': 0.001}	0,10262088
14	{'dropout_rate': 0.25, 'learning_rate': 0.01}	0,212993721
15	{'dropout_rate': 0.25, 'learning_rate': 0.1}	0,370062749
16	{'dropout_rate': 0.4, 'learning_rate': 1e-05}	0,60520487
17	{'dropout_rate': 0.4, 'learning_rate': 0.0001}	0,322259128
18	{'dropout_rate': 0.4, 'learning_rate': 0.001}	0,212809165
19	{'dropout_rate': 0.4, 'learning_rate': 0.01}	0,23606497
20	{'dropout_rate': 0.4, 'learning_rate': 0.1}	0,220930239
21	{'dropout_rate': 0.5, 'learning_rate': 1e-05}	0,723883351
22	{'dropout_rate': 0.5, 'learning_rate': 0.0001}	0,238449616
23	{'dropout_rate': 0.5, 'learning_rate': 0.001}	0,086378733
24	{'dropout_rate': 0.5, 'learning_rate': 0.01}	0,125876725
25	{'dropout_rate': 0.5, 'learning_rate': 0.1}	0,17349577

Análise de Resultados - Random Search

Hiperparâmetros Investigados:

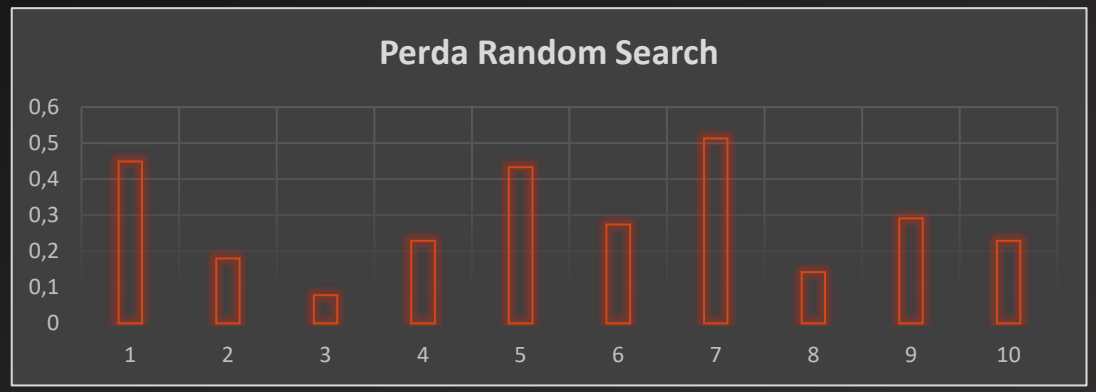
Taxas de aprendizado: 0.00001 a 0.1;

Taxas de dropout: 0.0 a 0.5;

10 iterações;

Conclusões:

Accuracy: 86.77%
Melhor taxa de aprendizado: 0.00049477721999320 62
Melhor taxa de dropout: 0.18104244959298804



Numero	Hiperparâmetros	Perda
1	{'dropout_rate': 0.031579100190464504, 'learning_rate': 3.79242450818784e-05}	0,449427823
2	{'dropout_rate': 0.38796513308386993, 'learning_rate': 0.00023374818007527835}	0,180509408
3	{'dropout_rate': 0.18104244959298804, 'learning_rate': 0.0004947772199932062}	0,078811367
4	{'dropout_rate': 0.3144270322421352, 'learning_rate': 0.005541871914094707}	0,228866736
5	{'dropout_rate': 0.11452316492913478, 'learning_rate': 8.649101093815958e-05}	0,432816525
6	{'dropout_rate': 0.4726716786188623, 'learning_rate': 0.0028395267348384303}	0,274270932
7	{'dropout_rate': 0.16987916629386127, 'learning_rate': 4.8708287086492195e-05}	0,512919893
8	{'dropout_rate': 0.22715297305018117, 'learning_rate': 0.001259452871896553}	0,142118851
9	{'dropout_rate': 0.18723842094751358, 'learning_rate': 0.01672217456163901}	0,29088225
10	{'dropout_rate': 0.42014021285476666, 'learning_rate': 0.0042554079239586935}	0,229235868

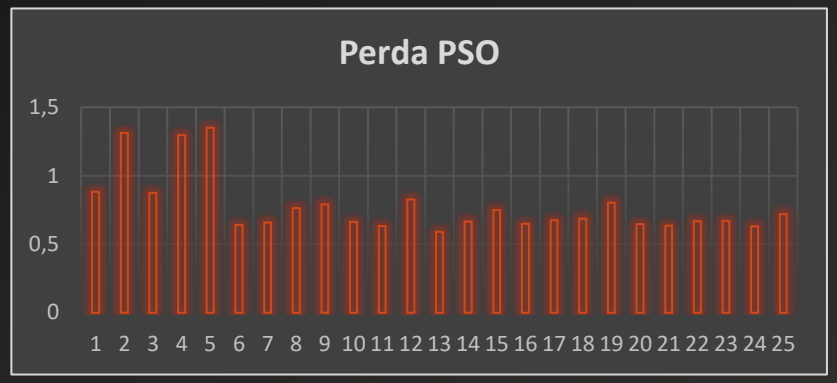
Análise de Resultados - PSO

Hiperparâmetros Investigados:

Taxas de aprendizado: 0.00001 a 0.1;
Taxas de dropout: 0.0 a 0.5;
5 partículas;
5 iterações;

Conclusões:

Acc: 79.46%
Melhor taxa de aprendizado: 0.00016697016593511955
Melhor taxa de dropout: 0.40144026943574496



Numero	Hiperparâmetros	Perda
1	[9.52776510e-05 1.71802231e-01]	0,881279051
2	[8.14390400e-07 1.04593423e-01]	1,311531544
3	[5.59232402e-05 1.66646774e-01]	0,872183502
4	[3.92025515e-06 2.03755155e-01]	1,29563868
5	[8.11282021e-06 4.00349337e-01]	1,35039103
6	[0.00013343 0.12715926]	0,639816105
7	[1.67781453e-04 2.44262440e-01]	0,657285035
8	[1.46062852e-04 3.17780025e-01]	0,763338029
9	[1.14014090e-04 4.98584864e-01]	0,792170823
10	[1.43830331e-04 2.26430594e-01]	0,6630885
11	[1.77762239e-04 4.46980578e-01]	0,632975459
12	[1.25752083e-04 3.07206856e-01]	0,825536728
13	[1.66970166e-04 4.01440269e-01]	0,590598285
14	[1.83795794e-04 3.44921246e-01]	0,664066672
15	[1.04945606e-04 4.04969207e-01]	0,748401701
16	[1.46243450e-04 3.25115788e-01]	0,648181796
17	[1.83847834e-04 3.20156273e-01]	0,674426138
18	[1.25786749e-04 4.36734490e-01]	0,686932504
19	[1.25657815e-04 1.81086989e-01]	0,802810013
20	[0.0001863 0.12162108]	0,64598763
21	[1.90331916e-04 2.37989411e-01]	0,634101808
22	[1.56810244e-04 2.63465562e-01]	0,667064965
23	[1.97421832e-04 4.06638194e-01]	0,669099867
24	[1.63390108e-04 4.59380631e-01]	0,630251408
25	[1.36057333e-04 2.54333485e-01]	0,717876256

Análise de Resultados – Modelo Simples

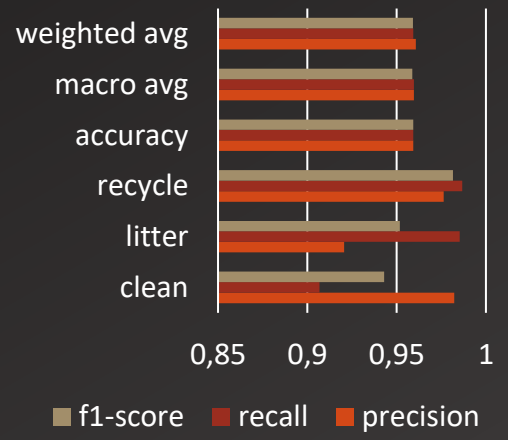
Hiperparâmetros Investigados:

Os melhores de entre os métodos PSO, Grid e Random Search, neste caso Random Search;

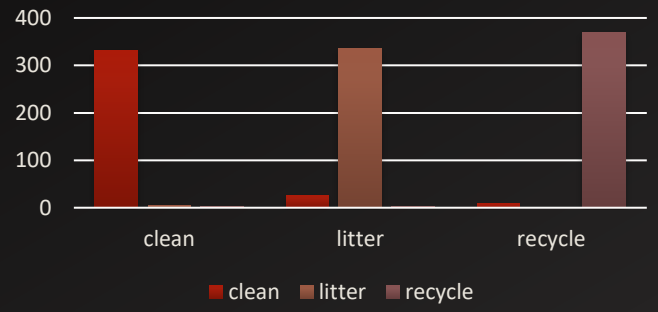
Learning_rate: 0.000495

Dropout_rate: 0.18

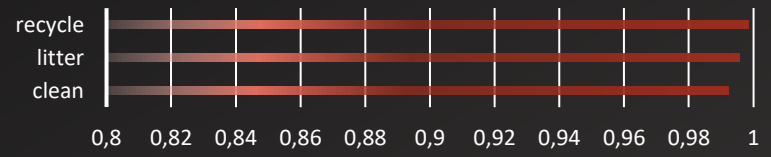
Relatório classificação - Simple



Matriz Confusão - Simple



AUC - SIMPLE



-	clean	litter	recycle
clean	331	26	8
litter	4	336	1
recycle	2	3	370

Classe	AUC
clean	0,992257213
litter	0,995690338
recycle	0,998534466

-	precision	recall	f1-score	support
clean	0,982195846	0,906849315	0,943019943	365
litter	0,920547945	0,985337243	0,95184136	341
recycle	0,976253298	0,986666667	0,981432361	375
accuracy	0,959296947	0,959296947	0,959296947	0,959296947
macro avg	0,959665696	0,959617742	0,958764555	1081
weighted avg	0,960687622	0,959296947	0,959127954	1081

Teste com o modelo treinado

Referências

Referências consultadas pela equipa, incluindo o uso do ChatGPT para assistência na elaboração deste relatório:

- Livros e Textos Acadêmicos:
  - Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). https://doi.org/10.1038/s41586-020-2649-2.
  - McKinney, W. et al., Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010).
  - Pedregosa et al., Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830 (2011).
  - Abadi, M., et al., TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems (2015). Software available from tensorflow.org.
  - Chollet, F., et al., Keras. (2015). https://keras.io.
  - Hunter, J.D., Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95 (2007).
  - Kingma, D.P., and Ba, J., Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG] (2014).
  - Clerc, M., Particle Swarm Optimization. ISTE (2006). ISBN: 1905209045.
  - Miranda, L., et al., PySwarms: a research toolkit for Particle Swarm Optimization in Python, Journal of Open Source Software, 3(21), 433 (2018). doi: 10.21105/joss.00433.
- Fichas Práticas da cadeira Inteligência Computacional, 2023/24;
- PDFs Teóricos da cadeira Inteligência Computacional, 2023/24;